Intrinsic Graphics
Invention Disclosure Form

Name of Person Submitting Form: Reuel Nash

Date Form Completed: 10/20/00

[redacted]

I. The Invention

1. Title of Invention: Method to efficiently use limited resources in a graphics device.

2. Describe the invention briefly (one paragraph or less).

This is a software architecture that provides efficient use of limited resources on Playstation 2 and other graphics hardware systems. Special buffering and synchronization techniques allow an application to run in a CPU-limited phase without causing undue idle time in graphics processors. The techniques allow simple applications to load texture images dynamically without considering memory allocations, placement, or synchronization.

3. Problem: State the problem which motivated or required a solution provided by the invention.

Graphical applications typically have some CPU-limited calculations (e.g. scene-graph traversal, physics, animation) that will generate data that is to be sent to the transformation processor and eventually to the pixel drawing processor. Typical double-buffered graphics systems cannot allow drawing to occur while the target frame buffer (front buffer) memory is being displayed, so they cause the application to wait until vertical retrace has begun. This results in significant CPU time being used after drawing could commence, so the transformation and pixel processors are idle.

Texture memory is limited. On game consoles, it's typical for a game application to need to reload texture memory several times while drawing a display frame. There are two constraints that come into play for loading textures: the texture image must be loaded into texture memory before it is used and no other texture image can replace a texture image until its use is complete. This is complicated by the structure of graphics system hardware that allows for asynchronous transformation and drawing processing and asynchronous texture image loading. Having the application perform synchronization operations can result in loss of processing time on the CPU, transformation processor, and pixel drawing processor and is quite complex for an application to implement. Applications would be trying to find out how long before a texture is needed should its loading begin.

4. Solution: Describe how the invention solves the problem and any other advantages of the invention.

The loss of processing time due to applications being forced to wait until vertical retrace to begin is solved with a combination of buffering and synchronization techniques. Application functions which generate graphics data call graphics library functions which save the data in a buffer for later transfer to the transformation processor and eventually to the drawing processor. Each buffer can have one or more predicate functions which must be satified before the transfer is started. Each system function that can cause a predicate function to be satisfied (typically interrupt service routines) checks to see if all of the current buffer's predicates are true. If so, the transfer is started. By making one of the predicate functions check to see if a specific frame index is current, the application can continue to run and generate graphical data while the buffer system holds it until the appropriate vertical retrace. Our implementation prevents the application from getting more that one frame ahead, although this is not necessary in general.

The texture loading problem is solved by making use of the same buffering and synchronization techniques along with a lazy texture memory deallocation scheme. Applications issue a sequence of commands to **load**, **use**, **draw**, and **unload** a texture image. The **load** command allocates texture memory and begins transfer of the image data if the texture loading path is not in use. The **use** command attaches a predicate function to the current graphics data buffers that requires that the texture be loaded before the buffer is transferred to the transformation processor. **Draw**ing commands can continue to be placed in buffers. The **unload** command puts a synchronization command into the buffer. An interrupt service routine is activated when the texture loading has completed. This routine validates the predicate for that texture and may start the transfer of graphics data if all predicates are satisfied. When the unload synchronization command causes an interrupt to occur, the texture is deallocated by the interrupt service routine. The load command will wait for all pending deallocations to occur if necessary to allocate memory for a new texture image.

5. a. Identify each *Intrinsic Graphics* product, design, service or business method that uses or may use the invention.

Intrinsic Alchemy for Playstation 2 uses this invention.

b. What are other possible uses for the invention? Please speculate reasonably on any additional uses that the invention may have, either by itself or in combination with other known, or as yet unknown, technology? For example, identify any *Intrinsic Graphics* competitor's product, design, service or business method that might be modified to use the invention.

Any graphics hardware could use these techniques.

6. Detailed Description: Drawings and Text.
Describe your invention and any variations of your invention. If possible, attach a specification, a manuscript, a research proposal, sketches, drawings, photographs, or any other materials that would assist in understanding the invention or in the drafting of figures or text in a patent application covering the invention.

A document I wrote about the texture loading method:

Low-level Support for Texture Mapping on PS2

Reuel Nash
August 10, 2000

Abstract

The near-optimal use of limited hardware resources available to game programmers is critical to the overall performance of the game and therefore to the success of the game. One such limited resource is texture memory. Many attempts have been made to virtualize this limited resource with less than favorable results. Programmers of real-time graphics applications have been known to work around "texture managers" in OpenGL to achieve repeatable results. This paper describes the internal functionality of the texture memory allocation and loading mechanisms provided under Alchemy for Playstation 2. This code has been demonstrated to support loading and drawing with 10 MB of texture per frame at 60 Hz.

Supported Formats

The code supports 32-, 24-, and 16-bit texture formats directly. 8- and 4-bit formats are supported with associated Color Look-Up Tables (CLUTs). The 8H, 4HH, and 4HL formats are not supported at this time due to the complexity of allocating them independently of the other formats.

Asynchronous Loading and Unloading

Texture images are loaded into texture memory along Path 3 (GIF->GS memory). The placement algorithm (described below) and synchronization techniques guarantee that no primitives can be using the target texture memory for drawing, so a texture download can begin whenever texture memory is available and the Path 3 DMA unit is idle. Drawing commands are put into relatively long FIFO buffers along with predicate infomation that prevents drawing DMA from starting until all textures used in each buffer have been loaded. This scheme allows the CPU to execute code as much as one frame ahead of the geometry currently being drawn and allows the texture loading to begin and finish before drawing commands which need the texture are sent to the GS. When all drawing commands for one texture have been placed into the DMA FIFO, the application or scene graph code will issue a request to relinquish the texture memory for that texture which results in a synchronization command being put into the FIFO buffer. When this synchronization command executes on the GS it will notify the memory allocator that all drawing with this texture is actually finished and the texture memory is available for loading other textures. In effect, all available texture memory is used as a staging buffer, with some textures being loaded, some available for actual drawing, and others being unloaded.

Synchronizing with GLL and Application code

Since no copy is made of the texture image before returning control of the processor to the application, the application cannot modify or free the texture image until the DMA is completed. The PSX2 code provides a query function to tell when a texture has actually completed the loading process.

Placement Algorithm

We use a linear, first-fit placement algorithm to determine the texture memory address for loading a texture. If a large enough contiguous space for the texture is available, it will be used. If no space is found, but there are outstanding requests to relinquish texture memory, then the code waits for the outstanding requests to finish. If no contiguous space is available after all outstanding requests have finished, then a failure indication is returned. No attempt to compact texture memory or automatically choose textures to remove is made. Appropriate response to this failure indication would be to relinquish more (perhaps all) textures and try to load the texture again.

Address Restrictions and Optimal Sizing

We have chosen to simplify the placement algorithm and memory allocator as described above. Two results of this simplification are some restriction on the starting address of a texture of a given size and the waste of some texture memory for textures of some particular sizes. Textures in the 32- and 24-bit formats will have a height that is promoted to at least half of the width of the texture and a width that is promoted to the height or 64 texels if the height is greater than 64 texels. For example, a 32-bit 8w X 32h texture will take up the memory of a 32x32 texture. A 32-bit 32w X 8h texture will take up the size of a 32x16 texture. The textures will be aligned to the natural alignment of the expanded texture size (e.g. a 32x32 texture will be aligned on a 1024 texel address boundary in the texture memory). Similar texture expansion and alignment restrictions exist for the other formats. A table of all possible sizes and their expanded allocation is in the appendix.

7. Identify the best way (or ways) of carrying out the invention that you know of that have not been mentioned above. For example, if the invention involves a product (hardware or software), what are the optimum structural or functional features? Of a composition, what are the optimum materials and proportions? If a process, what are the optimum conditions and parameters?

[redacted]

[redacted]